**OBJECT ORIENTED PROGRAMMING (OOP) PROJECT**

**Title: Car-Pooling System**

**Date Submission: 06/04/2025**

**Submitted To:** Dr. Deepak Kumar Sharma

**Prepared By:**

Abhinav Tyagi, 500122726

Varchasv Pandey, 500122631

Om Vishnu, 500124476



**School Of Computer Science**  
**UNIVERSITY OF PETROLEUM & ENERGY STUDIES, DEHRADUN-248007, Uttarakhand**

**Car-Pooling System**

**Abstract**

Car Pooling System This is a simple Java-based application that simulates a simplified version of a ride-sharing platform that exists in real life. It is an interface that helps to share transport means by allowing a user to book rides, compute fares, and manage the drivers through a console-based interface. It incorporates core features such as driver management, fare computation, and history keeping of bookings-all supported through file handling mechanisms. The system adopts consistent data storage through text files and makes use of object-oriented principles to organize functionalities in modular Java classes. The project is an actual implementation of how fundamental Java principles can be used to address real-world transport issues economically and efficiently.

**Main Functionalities of the Car-Pooling System**

1. Driver Registration

* Class: Driver
* Package: drivers
* Functionality: Allows a new driver to register with their name and current location.

Details:

* Method: public static void registerDriver()
  + Takes user input (name, location)
  + Automatically assigns a unique driver ID
  + Saves this info to drivers.txt
* Helper Method: private static int getNextDriverID()
  + Reads the existing file to find the next available ID.

Purpose: Ensures new drivers are added to the system and are available for future bookings.

2. Ride Booking (Customer-Side)

* Class: Booking
* Package: booking
* Functionality: Manages the full flow of booking a ride.

Details:

* Method: public int[] book()
  + Uses Pick class to select pickup and drop points.
  + Calculates distance and fare using FareCalculator.
  + Assigns the nearest driver using Driver.assignDriver().
  + Updates driver location to drop point with Driver.updateDriverLocation().
  + Saves booking history to booking\_history.txt.

Purpose: Automates booking with real-time driver assignment and fare calculation.

3. Fare & ETA Calculation

* Class: FareCalculator
* Package: booking
* Functionality: Calculates total fare and estimated arrival time based on distance.

Details:

* Constants:
  + COST\_PER\_KM = 12
  + AVG\_SPEED = 40 km/h
* Methods:
  + public static int calculateFare(int dist) → fare = distance × cost/km
  + public static int calculateETA(int dist) → ETA = distance ÷ speed × 60

Purpose: Provides transparent pricing and realistic arrival estimates.

4. Driver Assignment & Management

* Class: Driver
* Package: drivers
* Functionality: Finds the closest driver and updates their location after booking.

Details:

* Method: public static String[] assignDriver(String pickupLocation)
  + Reads from drivers.txt
  + Finds the closest driver based on location index difference
* Method: public static void updateDriverLocation(String driverID, String newLocation)
  + Updates driver’s location after drop by rewriting the file

Purpose: Maintains real-time driver availability and location tracking.

5. Route and Distance Selection

* Class: Pick
* Package: routes
* Functionality: Lets user choose pickup and drop locations; calculates the distance.

Details:

* Method: public int choice()
  + Prompts user for pickup and drop points
  + Uses predefined distances: [9, 11, 7, 13] km between spots
  + Returns total distance
* Other Methods:
  + getPickupLocation() and getDropLocation() → Return location names for display

Purpose: Simplifies route input and feeds data into booking/fare logic.

6. Main Program Flow

* Class: Ques1
* Package: main
* Functionality: Main entry point for both drivers and customers.

Details:

* Method: public static void main(String[] args)
  + Asks user type: Driver or Customer
  + If Driver → calls Driver.registerDriver()
  + If Customer → calls Booking.book() and displays fare/ETA

Purpose: Launches the app and directs users based on their role.

**Complete Code**

1. Driver Assignment

Code:

package drivers;

import java.io.\*;

import java.util.\*;

public class Driver

{

static String[] locations = {"Bidholi", "Prem Nagar", "Clock Tower", "Bus Stop", "Train Station"};

public static void registerDriver()

{

Scanner s = new Scanner(System.in);

System.out.println("Enter your name:");

String name = s.nextLine();

System.out.println("Select your current location:");

for (int i = 0; i < locations.length; i++)

{

System.out.println((i + 1) + ". " + locations[i]);

}

int locIndex = s.nextInt() - 1;

String location = locations[locIndex];

int driverID = getNextDriverID();

try

{

FileWriter fw = new FileWriter("drivers.txt", true);

fw.write(driverID + "," + name + "," + location + "\n");

fw.close();

System.out.println("Driver registered successfully. Your Driver ID is: " + driverID);

}

catch (IOException e)

{

System.out.println("Error saving driver info.");

}

}

private static int getNextDriverID()

{

int id = 1;

try

{

BufferedReader br = new BufferedReader(new FileReader("drivers.txt"));

String line;

while ((line = br.readLine()) != null)

{

String[] parts = line.split(",");

if (parts.length > 0)

{

int lastID = Integer.parseInt(parts[0]);

if (lastID >= id)

{

id = lastID + 1;

}

}

}

br.close();

}

catch (FileNotFoundException e)

{

}

catch (IOException e)

{

e.printStackTrace();

}

return id;

}

public static String[] assignDriver(String pickupLocation)

{

String[] closestDriverData = null;

int minDistance = Integer.MAX\_VALUE;

try

{

BufferedReader br = new BufferedReader(new FileReader("drivers.txt"));

String line;

while ((line = br.readLine()) != null)

{

String[] parts = line.split(",");

if (parts.length >= 3)

{

String driverID = parts[0];

String name = parts[1];

String location = parts[2];

int dist = Math.abs(getLocationIndex(location) - getLocationIndex(pickupLocation));

if (dist < minDistance)

{

minDistance = dist;

closestDriverData = parts;

}

}

}

br.close();

if (closestDriverData != null)

{

System.out.println("Driver Found!");

}

}

catch (IOException e)

{

System.out.println("Error finding drivers.");

}

return closestDriverData;

}

public static void updateDriverLocation(String driverID, String newLocation)

{

try

{

File file = new File("drivers.txt");

File tempFile = new File("drivers\_temp.txt");

BufferedReader reader = new BufferedReader(new FileReader(file));

BufferedWriter writer = new BufferedWriter(new FileWriter(tempFile));

String line;

while ((line = reader.readLine()) != null)

{

String[] parts = line.split(",");

if (parts.length >= 3 && parts[0].equals(driverID))

{

writer.write(parts[0] + "," + parts[1] + "," + newLocation + "\n");

}

else

{

writer.write(line + "\n");

}

}

writer.close();

reader.close();

file.delete();

tempFile.renameTo(file);

}

catch (IOException e)

{

System.out.println("Error updating driver location.");

}

}

private static int getLocationIndex(String locName)

{

for (int i = 0; i < locations.length; i++)

{

if (locations[i].equalsIgnoreCase(locName))

{

return i;

}

}

return -1;

}

}

2. Ride Booking

Code:

package booking;

import java.io.FileWriter;

import java.io.IOException;

import routes.Pick;

import drivers.Driver;

public class Booking

{

public int[] book()

{

Pick p = new Pick();

int dist = p.choice();

String pickupLoc = p.getPickupLocation();

String dropLoc = p.getDropLocation();

String[] assignedDriver = Driver.assignDriver(pickupLoc);

String driverName = (assignedDriver != null) ? assignedDriver[1] : "No driver available";

String driverID = (assignedDriver != null) ? assignedDriver[0] : null;

if (driverID != null)

{

Driver.updateDriverLocation(driverID, dropLoc);

}

int fare = FareCalculator.calculateFare(dist);

int eta = FareCalculator.calculateETA(dist);

int[] outp = {fare, eta};

try

{

FileWriter writer = new FileWriter("booking\_history.txt", true);

writer.write("Pickup: " + pickupLoc + ", Drop: " + dropLoc + ", Distance: " + dist + " km, Fare: ₹" + fare + ", ETA: " + eta + " min, Driver: " + driverName + "\n");

writer.close();

}

catch (IOException e)

{

System.out.println("An error occurred while saving booking history.");

}

System.out.println("Driver Name : " + driverName);

return outp;

}

}

3. Fare & ETA Calculation

Code:

package booking;

public class FareCalculator

{

static int COST\_PER\_KM = 12;

static int AVG\_SPEED = 40;

public static int calculateFare(int dist)

{

return dist \* COST\_PER\_KM;

}

public static int calculateETA(int dist)

{

return (int)(((double) dist / AVG\_SPEED) \* 60);

}

}

5. Route and Distance Selection

Code:

package routes;

import java.util.Scanner;

public class Pick {

static String[] locations = {"Bidholi", "Prem Nagar", "Clock Tower", "Bus Stop", "Train Station"};

Scanner s = new Scanner(System.in);

int pickup;

int drop;

public int choice()

{

System.out.println("The routes are listed below : \n\tBidholi - Prem Nagar\n\tPrem Nagar - Clock Tower\n\tClock Tower - Bus Stop\n\tBus Stop - Train Station");

System.out.println("The spots are numbered as following :\n\t\t1 Bidholi\n\t\t2 Prem Nagar\n\t\t3 Clock Tower\n\t\t4 Bus Stop\n\t\t5 Train Station");

int[] dist = {9, 11, 7, 13};

System.out.println("Enter the pickup point : ");

pickup = s.nextInt();

System.out.println("Enter the drop point : ");

drop = s.nextInt();

if (pickup < 1 || pickup > 5 || drop < 1 || drop > 5)

{

System.out.println("Invalid pickup or drop point!");

return 0;

}

if (pickup == drop)

{

System.out.println("You can walk");

return 0;

}

if (pickup > drop)

{

int temp = pickup;

pickup = drop;

drop = temp;

}

int check = 0;

for (int i = pickup - 1; i < drop - 1; i++)

{

check += dist[i];

}

return check;

}

public int getPickup()

{

return pickup;

}

public int getDrop()

{

return drop;

}

public String getPickupLocation()

{

return locations[pickup - 1];

}

public String getDropLocation()

{

return locations[drop - 1];

}

}

6. Main Program Flow

Code:

public class Ques1

{

public static void main(String[] args)

{

Scanner s = new Scanner(System.in);

System.out.println("Welcome to the Car Pooling System!");

System.out.println("Are you a:\n1. Customer\n2. Driver");

int userType = s.nextInt();

if (userType == 2)

{

Driver.registerDriver();

}

else

{

Booking b = new Booking();

int[] info = b.book();

System.out.println("Your ride is booked. Rider will arrive soon.");

System.out.println("Estimated time of arrival (ETA): " + info[1] + " minutes");

System.out.println("Total fare: ₹" + info[0]);

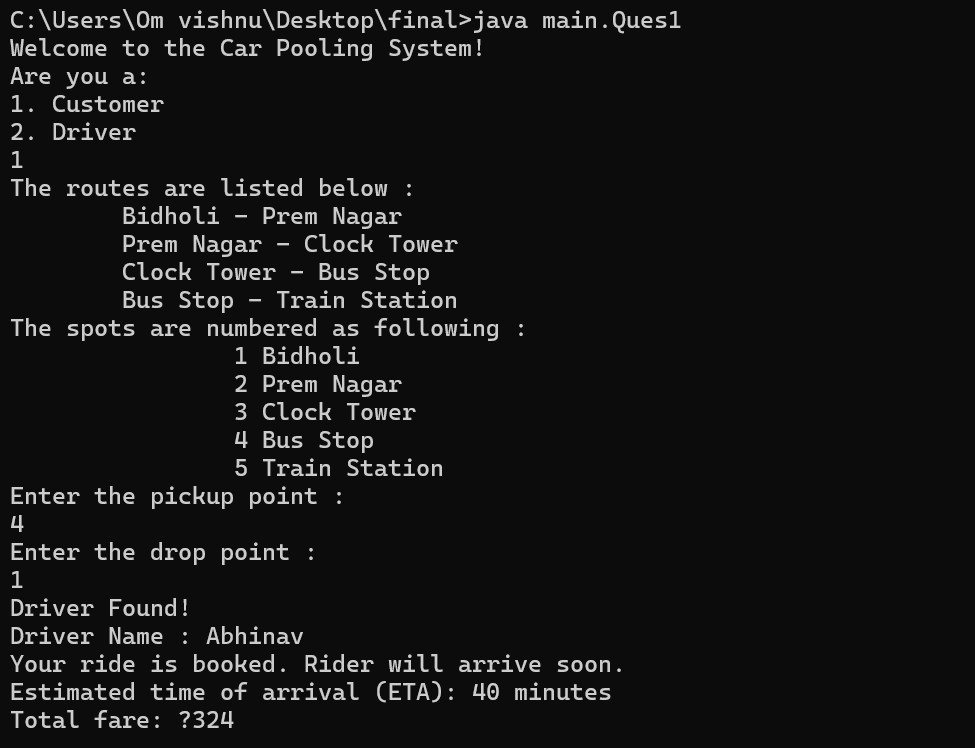
}

}

}

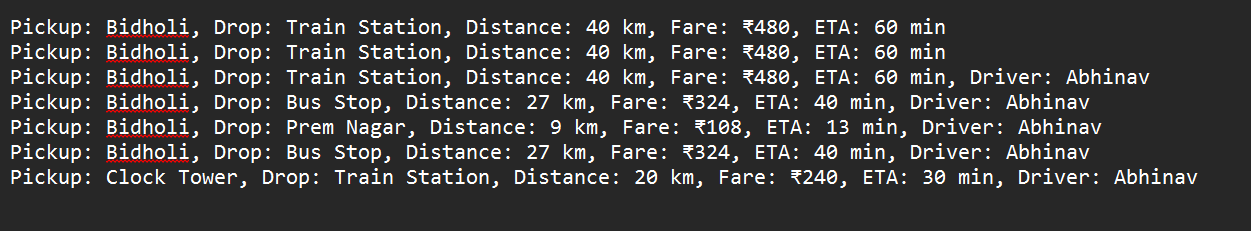
**Outputs:**

Ride Booking



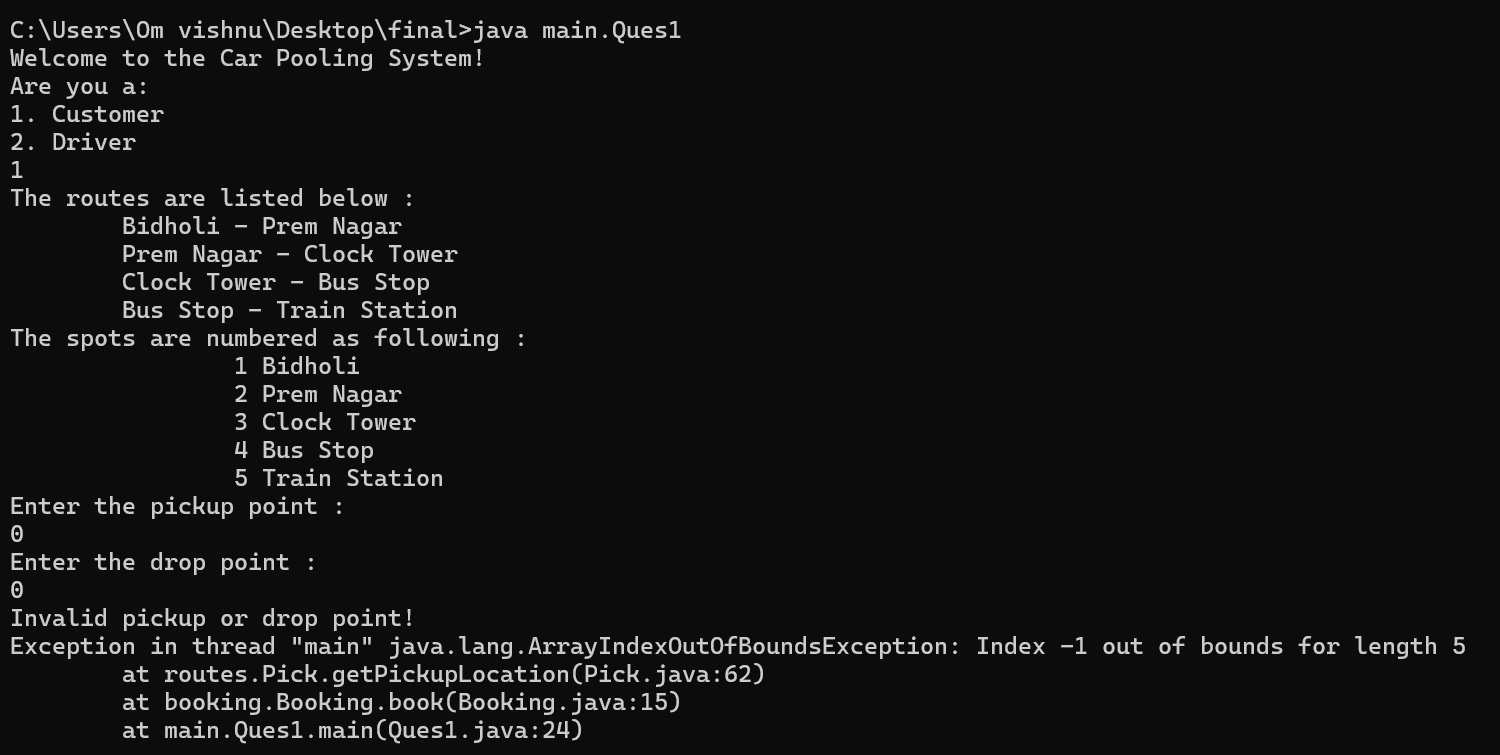
In this output, the user selects the Customer option, enters pickup and drop points (Bus Stop to Bidholi), and the system calculates the distance (27 km), fare (₹324), and estimated time of arrival (40 minutes) using methods from the Pick and FareCalculator classes. It then assigns the nearest available driver using Driver.assignDriver() and updates their location after the booking. Finally, the booking details are recorded, and the driver’s name, ETA, and fare are displayed to the user, showcasing the core functionalities of route handling, fare estimation, driver assignment, and booking confirmation in the car-pooling system.

File Handling



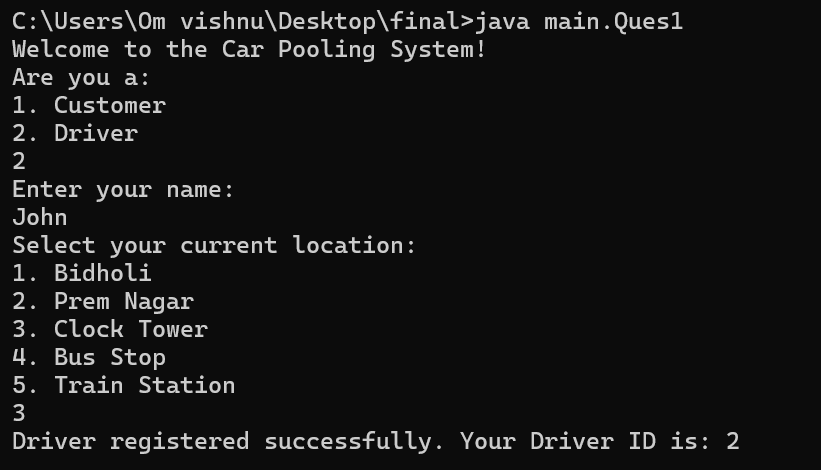
This output display the use of file handling, after the ride has been booked the history of that ride is stored in a text file it is done in the book() method of the Booking class using the FileWriter class. Using this we can store the customer’s ride detail like pickup, drop, distance, fare, ETA and driver’s name.

Exceptional handling



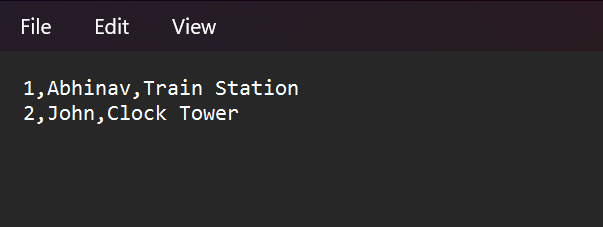
In this output, the user selects an invalid pickup and drop point (0), which causes the program to throw an ArrayIndexOutOfBoundsException with index -1, as Java arrays are zero-indexed and the location list starts from 1 to 5. The error occurs in the getPickupLocation() method in the Pick class, and is traced through the book() method in Booking and the main() method. Although the program displays a message "Invalid pickup or drop point!", it lacks proper **exception handling** like a try-catch block to prevent the crash. Input validation should be implemented to ensure user entries fall within the allowed range before accessing the array.

Driver Registration



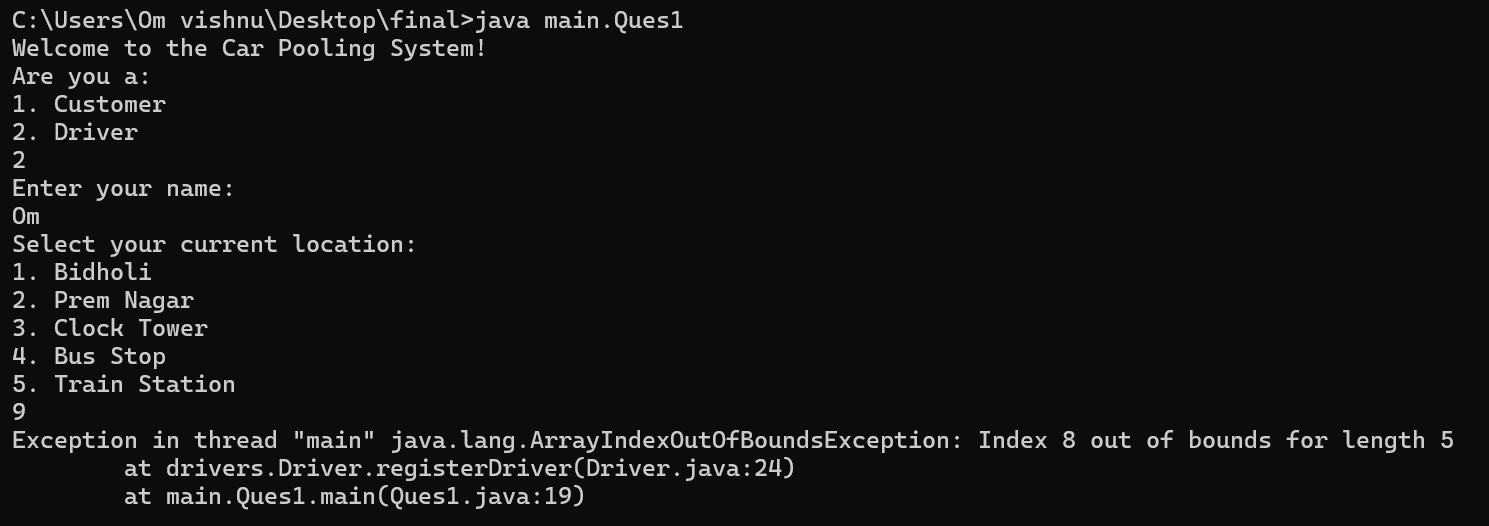
In this output, the user selects the **Driver** option, enters their name ("John"), and chooses their current location (Clock Tower). The program then registers the driver using the registerDriver() method from the Driver class. This method collects input, assigns a unique driver ID using getNextDriverID().

File Handling



The driver's details (ID, name, and location) are stored in the drivers.txt file using FileWriter.

Exceptional Handling



In this output, the program throws an ArrayIndexOutOfBoundsException because the user entered an invalid location choice (9), which exceeds the valid index range (1 to 5). The error occurs in the registerDriver() method of the Driver class at line 24, where the selected location is accessed from an array without checking if the index is within bounds. This highlights the need for exception handling using a try-catch block or input validation to prevent runtime crashes and prompt the user for valid input instead of terminating the program.